**Unix Interview Questions on Awk Command**

Vijay Bhaskar11/15/2011[2 Comments](https://www.folkstalk.com/2011/11/unix-interview-questions-on-awk-command.html#comments)

* <https://www.folkstalk.com/2011/11/unix-interview-questions-on-awk-command.html>

Awk is powerful tool in Unix. Awk is an excellent tool for processing the files which have data arranged in rows and columns format. It is a good filter and report writer.  
1. How to run awk command specified in a file?  
awk -f filename  
  
2. Write a command to print the squares of numbers from 1 to 10 using awk command  
awk 'BEGIN { for(i=1;i<=10;i++) {print "square of",i,"is",i\*i;}}'  
  
3. Write a command to find the sum of bytes (size of file) of all files in a directory.  
ls -l | awk 'BEGIN {sum=0} {sum = sum + $5} END {print sum}'  
  
4. In the text file, some lines are delimited by colon and some are delimited by space. Write a command to print the third field of each line.  
  
awk '{ if( $0 ~ /:/ ) { FS=":"; } else { FS =" "; } print $3 }' filename  
  
5. Write a command to print the line number before each line?  
awk '{print NR, $0}' filename  
  
6. Write a command to print the second and third line of a file without using NR.  
awk 'BEGIN {RS="";FS="\n"} {print $2,$3}' filename  
  
7. Write a command to print zero byte size files?  
ls -l | awk '/^-/ {if ($5 !=0 ) print $9 }'  
  
8. Write a command to rename the files in a directory with "\_new" as postfix?  
ls -F | awk '{print "mv "$1" "$1".new"}' | sh

**Top Examples of Awk Command in Unix**

Vijay Bhaskar12/09/2011

* [Facebook](javascript:;)
* [Twitter](javascript:;)

Awk is one of the most powerful tools in Unix used for processing the rows and columns in a file. Awk has built in string functions and associative arrays. Awk supports most of the operators, conditional blocks, and loops available in C language.  
  
One of the good things is that you can convert Awk scripts into Perl scripts using a2p utility.  
  
**The basic syntax of AWK:**

awk 'BEGIN {start\_action} {action} END {stop\_action}' filename

Here the actions in the begin block are performed before processing the file and the actions in the end block are performed after processing the file. The rest of the actions are performed while processing the file.  
  
**Examples:**  
  
Create a file input\_file with the following data. This file can be easily created using the output of ls -l.

-rw-r--r-- 1 center center  0 Dec  8 21:39 p1

-rw-r--r-- 1 center center 17 Dec  8 21:15 t1

-rw-r--r-- 1 center center 26 Dec  8 21:38 t2

-rw-r--r-- 1 center center 25 Dec  8 21:38 t3

-rw-r--r-- 1 center center 43 Dec  8 21:39 t4

-rw-r--r-- 1 center center 48 Dec  8 21:39 t5

From the data, you can observe that this file has rows and columns. The rows are separated by a new line character and the columns are separated by a space characters. We will use this file as the input for the examples discussed here.  
  
**1.** awk '{print $1}' input\_file  
  
Here $1 has a meaning. $1, $2, $3... represents the first, second, third columns... in a row respectively. This awk command will print the first column in each row as shown below.

-rw-r--r--

-rw-r--r--

-rw-r--r--

-rw-r--r--

-rw-r--r--

-rw-r--r--

To print the 4th and 6th columns in a file use awk '{print $4,$5}' input\_file  
  
Here the Begin and End blocks are not used in awk. So, the print command will be executed for each row it reads from the file. In the next example we will see how to use the Begin and End blocks.  
  
**2.** awk 'BEGIN {sum=0} {sum=sum+$5} END {print sum}' input\_file

This will prints the sum of the value in the 5th column. In the Begin block the variable sum is assigned with value 0. In the next block the value of 5th column is added to the sum variable. This addition of the 5th column to the sum variable repeats for every row it processed. When all the rows are processed the sum variable will hold the sum of the values in the 5th column. This value is printed in the End block.  
  
**3.** In this example we will see how to execute the awk script written in a file. Create a file sum\_column and paste the below script in that file

#!/usr/bin/awk -f

BEGIN {sum=0}

{sum=sum+$5}

END {print sum}

Now execute the the script using awk command as  
  
awk -f sum\_column input\_file.  
  
This will run the script in sum\_column file and displays the sum of the 5th column in the input\_file.  
  
**4.** awk '{ if($9 == "t4") print $0;}' input\_file  
  
This awk command checks for the string "t4" in the 9th column and if it finds a match then it will print the entire line. The output of this awk command is

-rw-r--r-- 1 pcenter pcenter 43 Dec  8 21:39 t4

**5.** awk 'BEGIN { for(i=1;i<=5;i++) print "square of", i, "is",i\*i; }'  
  
This will print the squares of first numbers from 1 to 5. The output of the command is

square of 1 is 1

square of 2 is 4

square of 3 is 9

square of 4 is 16

square of 5 is 25

Notice that the syntax of “if” and “for” are similar to the C language.  
  
**Awk Built in Variables:**  
  
You have already seen $0, $1, $2... which prints the entire line, first column, second column... respectively. Now we will see other built in variables with examples.  
  
**FS** - Input field separator variable:  
  
So far, we have seen the fields separted by a space character. By default Awk assumes that fields in a file are separted by space characters. If the fields in the file are separted by any other character, we can use the FS variable to tell about the delimiter.  
  
**6.** awk 'BEGIN {FS=":"} {print $2}' input\_file  
OR  
awk -F: '{print $2}' input\_file  
  
This will print the result as

39 p1

15 t1

38 t2

38 t3

39 t4

39 t5

**OFS** - Output field separator variable:  
  
By default whenever we printed the fields using the print statement the fields are displayed with space character as delimiter. For example  
  
**7.** awk '{print $4,$5}' input\_file  
  
The output of this command will be

center 0

center 17

center 26

center 25

center 43

center 48

We can change this default behavior using the OFS variable as  
  
awk 'BEGIN {OFS=":"} {print $4,$5}' input\_file

center:0

center:17

center:26

center:25

center:43

center:48

Note: print $4,$5 and print $4$5 will not work the same way. The first one displays the output with space as delimiter. The second one displays the output without any delimiter.  
  
**NF** - Number of fileds variable:  
  
The NF can be used to know the number of fields in line  
  
**8.** awk '{print NF}' input\_file  
This will display the number of columns in each row.  
  
**NR** - number of records variable:  
The NR can be used to know the line number or count of lines in a file.  
  
**9.** awk '{print NR}' input\_file  
This will display the line numbers from 1.  
  
**10.** awk 'END {print NR}' input\_file  
This will display the total number of lines in the file.  
  
**String functions in Awk:**  
Some of the string functions in awk are:  
  
index(string,search)  
length(string)  
split(string,array,separator)  
substr(string,position)  
substr(string,position,max)  
tolower(string)  
toupper(string)  
  
**Advanced Examples:**  
  
**1.** Filtering lines using Awk split function  
  
The awk split function splits a string into an array using the delimiter.  
  
The syntax of split function is  
split(string, array, delimiter)  
  
Now we will see how to filter the lines using the split function with an example.  
  
The input "file.txt" contains the data in the following format

1 U,N,UNIX,000

2 N,P,SHELL,111

3 I,M,UNIX,222

4 X,Y,BASH,333

5 P,R,SCRIPT,444

Required output: Now we have to print only the lines in which whose 2nd field has the string "UNIX" as the 3rd field( The 2nd filed in the line is separated by comma delimiter ).  
The ouptut is:

1 U,N,UNIX,000

3 I,M,UNIX,222

The awk command for getting the output is:

awk '{

split($2,arr,",");

if(arr[3] == "UNIX")

print $0

} ' file.txt  
9. Write a command to print the fields in a text file in reverse order?  
awk 'BEGIN {ORS=""} { for(i=NF;i>0;i--) print $i," "; print "\n"}' filename  
  
10. Write a command to find the total number of lines in a file without using NR  
awk 'BEGIN {sum=0} {sum=sum+1} END {print sum}' filename  
  
Another way to print the number of lines is by using the NR. The command is  
awk 'END{print NR}' filename

<https://www.folkstalk.com/2011/12/101-examples-of-using-find-command-in.html>

Find Command

Vijay Bhaskar12/26/2011[13 Comments](https://www.folkstalk.com/2011/12/101-examples-of-using-find-command-in.html#comments)

Find is one of the powerful utility of Unix (or Linux) used for searching the files in a directory hierarchy. The syntax of find command is

find [pathnames] [conditions]

Let see some practical exercises on using find command.  
  
**1.** How to run the last executed find command?

!find

This will execute the last find command. It also displays the last find command executed along with the result on the terminal.  
  
**2.** How to find for a file using name?

find -name "sum.java"

./bkp/sum.java

./sum.java

This will find all the files with name "sum.java" in the current directory and sub-directories.  
  
**3.** How to find for files using name and ignoring case?

find -iname "sum.java"

./SUM.java

./bkp/sum.java

./sum.java

This will find all the files with name "sum.java" while ignoring the case in the current directory and sub-directories.  
  
**4.** How to find for a file in the current directory only?

find -maxdepth 1 -name "sum.java"

./sum.java  
This will find for the file "sum.java" in the current directory only  
**5.** How to find for files containing a specific word in its name?

find -name "\*java\*"

./SUM.java

./bkp/sum.java

./sum.java

./multiply.java  
It displayed all the files which have the word "java" in the filename  
**6.** How to find for files in a specific directory?

find /etc -name "\*java\*"  
This will look for the files in the /etc directory with "java" in the filename  
**7.** How to find the files whose name are not "sum.java"?

find -not -name "sum.java"

.

./SUM.java

./bkp

./multiply.java

This is like inverting the match. It prints all the files except the given file "sum.java".  
  
**8.** How to limit the file searches to specific directories?

find -name "sum.java"

./tmp/sum.java

./bkp/var/tmp/files/sum.java

./bkp/var/tmp/sum.java

./bkp/var/sum.java

./bkp/sum.java

./sum.java

You can see here the find command displayed all the files with name "sum.java" in the current directory and sub-directories.  
  
**a.** How to print the files in the current directory and one level down to the current directory?

find -maxdepth 2 -name "sum.java"

./tmp/sum.java

./bkp/sum.java

./sum.java

**b.** How to print the files in the current directory and two levels down to the current directory?

find -maxdepth 3 -name "sum.java"

./tmp/sum.java

./bkp/var/sum.java

./bkp/sum.java

./sum.java

**c.** How to print the files in the subdirectories between level 1 and 4?

find -mindepth 2 -maxdepth 5 -name "sum.java"

./tmp/sum.java

./bkp/var/tmp/files/sum.java

./bkp/var/tmp/sum.java

./bkp/var/sum.java

./bkp/sum.java  
**9.** How to find the empty files in a directory?

find . -maxdepth 1 -empty

./empty\_file  
**10.** How to find the largest file in the current directory and sub directories

find . -type f -exec ls -s {} \; | sort -n -r | head -1  
The find command "find . -type f -exec ls -s {} \;" will list all the files along with the size of the file. Then the sort command will sort the files based on the size. The head command will pick only the first line from the output of sort.  
**11.** How to find the smallest file in the current directory and sub directories

find . -type f -exec ls -s {} \; | sort -n -r | tail -1  
Another method using find is

find . -type f -exec ls -s {} \; | sort -n | head -1  
**12.** How to find files based on the file type?  
**a.** Finding socket files

find . -type s  
**b.** Finding directories

find . -type d  
**c.** Finding hidden directories

find -type d -name ".\*"  
**d.** Finding regular files

find . -type f  
**e.** Finding hidden files

find . -type f -name ".\*"  
**13.** How to find files based on the size?  
**a.** Finding files whose size is exactly 10M

find . -size 10M  
**b.** Finding files larger than 10M size

find . -size +10M  
**c.** Finding files smaller than 10M size

find . -size -10M  
**14.** How to find the files which are modified after the modification of a give file.

find -newer "sum.java"  
This will display all the files which are modified after the file "sum.java"  
**15.** Display the files which are accessed after the modification of a give file.

find -anewer "sum.java"  
**16.** Display the files which are changed after the modification of a give file.

find -cnewer "sum.java"  
**17.** How to find the files based on the file permissions?

find . -perm 777  
This will display the files which have read, write, and execute permissions. To know the permissions of files and directories use the command "ls -l".  
  
**18.** Find the files which are modified within 30 minutes.

find . -mmin -30  
**19.** Find the files which are modified within 1 day.

find . -mtime -1  
**20.** How to find the files which are modified 30 minutes back

find . -not -mmin -30  
**21.** How to find the files which are modified 1 day back.

find . -not -mtime -1  
**22.** Print the files which are accessed within 1 hour.

find . -amin -60  
**23.** Print the files which are accessed within 1 day.

find . -atime -1  
**24.** Display the files which are changed within 2 hours.

find . -cmin -120  
**25.** Display the files which are changed within 2 days.

find . -ctime -2  
**26.** How to find the files which are created between two files.

find . -cnewer f1 -and ! -cnewer f2  
So far we have just find the files and displayed on the terminal. Now we will see how to perform some operations on the files.  
**1.** How to find the permissions of the files which contain the name "java"?

find -name "\*java\*"|xargs ls -l  
Alternate method is

find -name "\*java\*" -exec ls -l {} \;  
**2.** Find the files which have the name "java" in it and then display only the files which have "class" word in them?

find -name "\*java\*" -exec grep -H class {} \;  
**3.** How to remove files which contain the name "java".

find -name "\*java\*" -exec rm -r {} \;

# Grep Command in Unix and Linux Examples

<https://www.folkstalk.com/2012/01/grep-command-in-unix-examples.html>

Grep is the frequently used command in Unix (or Linux). Most of us use grep just for finding the words in a file. The power of grep comes with using its options and regular expressions. You can analyze large sets of log files with the help of grep command.  
  
Grep stands for Global search for Regular Expressions and Print.  
  
**The basic syntax of grep command is**  
grep [options] pattern [list of files]  
  
Let see some practical examples on grep command.  
  
**1.** Running the last executed grep command  
This saves a lot of time if you are executing the same command again and again.

!grep

This displays the last executed grep command and also prints the result set of the command on the terminal.  
  
**2.** Search for a string in a file  
  
This is the basic usage of grep command. It searches for the given string in the specified file.

grep "Error" logfile.txt

This searches for the string "Error" in the log file and prints all the lines that has the word "Error".  
**3.** Searching for a string in multiple files.

grep "string" file1 file2

grep "string" file\_pattern

This is also the basic usage of the grep command. You can manually specify the list of files you want to search or you can specify a file pattern (use regular expressions) to search for.  
**4.** Case insensitive search  
The -i option enables to search for a string case insensitively in the give file. It matches the words like "UNIX", "Unix", "unix".

grep -i "UNix" file.txt  
**5.** Specifying the search string as a regular expression pattern.

grep "^[0-9].\*" file.txt

This will search for the lines which starts with a number. Regular expressions is huge topic and I am not covering it here. This example is just for providing the usage of regular expressions.  
  
**6.** Checking for the whole words in a file.  
  
By default, grep matches the given string/pattern even if it found as a substring in a file. The -w option to grep makes it match only the whole words.

grep -w "world" file.txt

**7.** Displaying the lines before the match.  
Some times, if you are searching for an error in a log file; it is always good to know the lines around the error lines to know the cause of the error.

grep -B 2 "Error" file.txt

This will prints the matched lines along with the two lines before the matched lines.  
  
**8.** Displaying the lines after the match.

grep -A 3 "Error" file.txt

This will display the matched lines along with the three lines after the matched lines.  
  
**9.** Displaying the lines around the match

grep -C 5 "Error" file.txt

This will display the matched lines and also five lines before and after the matched lines.  
  
**10.** Searching for a sting in all files recursively  
You can search for a string in all the files under the current directory and sub-directories with the help -r option.

grep -r "string" \*

**11.** Inverting the pattern match  
You can display the lines that are not matched with the specified search sting pattern using the -v option.

grep -v "string" file.txt  
**12.** Displaying the non-empty lines  
You can remove the blank lines using the grep command.

grep -v "^$" file.txt  
**13.** Displaying the count of number of matches.  
We can find the number of lines that matches the given string/pattern

grep -c "sting" file.txt  
**14.** Display the file names that matches the pattern.  
We can just display the files that contains the given string/pattern.

grep -l "string" \*  
**15.** Display the file names that do not contain the pattern.  
We can display the files which do not contain the matched string/pattern.

grep -L "string" \*  
**16.** Displaying only the matched pattern.  
By default, grep displays the entire line which has the matched string. We can make the grep to display only the matched string by using the -o option.

grep -o "string" file.txt  
**17.** Displaying the line numbers.  
We can make the grep command to display the position of the line which contains the matched string in a file using the -n option

grep -n "string" file.txt  
**18.** Displaying the position of the matched string in the line  
The -b option allows the grep command to display the character position of the matched string in a file.

grep -o -b "string" file.txt  
**19.** Matching the lines that start with a string  
The ^ regular expression pattern specifies the start of a line. This can be used in grep to match the lines which start with the given string or pattern.

grep "^start" file.txt

**20.** Matching the lines that end with a string  
  
The $ regular expression pattern specifies the end of a line. This can be used in grep to match the lines which end with the given string or pattern.

grep "end$" file.txt